

4000-15800

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
APPLICATION FOR UNITED STATES LETTERS PATENT

ENTERPRISE NAMING SERVICE SYSTEM AND METHOD

By:

Omar Dewan
8501 W 127 Circle
Overland Park, KS 66213
Citizenship: USA

Shrikant Jannu
15347 S. Seminole Drive
Olathe, KS 66062
Citizenship: USA

Dora Potluri
3800 Skyline Drive
Plano, TX 75025
Citizenship: USA

Enterprise Naming Service System and Method

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] None.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not applicable.

REFERENCE TO A MICROFICHE APPENDIX

[0003] Not applicable.

FIELD OF THE INVENTION

[0004] The present invention is directed to methods and apparatuses for name services in distributed object systems, and more particularly, but not by way of limitation, to methods and apparatuses for implementing a unified name service.

BACKGROUND OF THE INVENTION

[0005] Enterprise software systems may comprise many independent computer programs, applications, modules, or components. These applications may execute in a distributed manner on several different computers. Applications often require the services provided by other applications. A first application requesting a service from a second application may be said to act in a client role while the second application may be said to act in a server role. The first application may provide services to yet other applications and

may in that case act in a server role. The intercommunicating applications may be executing on computers located far apart and in different divisions of the company.

[0006] The address or location of these services and servers may be maintained, such as by hard-coding, in client modules. When a server is relocated, perhaps to a different computer system to distribute loads evenly among multiple computer systems, the code on the client module must be changed to employ the new address of the server. When the client code is changed it must be installed and brought into service, in some cases interrupting operations, if only temporarily. The address or location of these services and servers may alternately be provided as a file entry, as in a configuration file or an initialization file. In this case, when a server is relocated the configuration or initialization files must be updated to reflect the new location of the service and or server.

SUMMARY OF THE INVENTION

[0007] The present embodiment provides a naming service for locating a service in an enterprise. The naming service comprising a binding module to associate a first service with a location of an interface maintaining a reference to the first service, the binding module further operable to associate a second service with a location of the second service. The naming service further comprising a look-up module operative to provide the location of the interface in response to a request by an application for the first service, the look-up module further operable to provide the location of the second service in response to a request by a second application.

[0008] In one embodiment of the naming service, the binder module is operable to associate the name of a java service object to the location of a java naming and directory interface having a reference to the java service object. A name look-up module is operable to provide a requesting application with the remote java naming and directory service. The location information, for example may contain the provider url, initial context factory, the associated java naming and directory interface and/or the full home interface name.

[0009] In another embodiment, the binder module is operable to associate the name of a CORBA object to an address or reference to the CORBA object, and a name look-up module is operable to use the association to provide the address or reference of the CORBA object to an application which has requested the look-up information of the CORBA object. The application may then use this CORBA object address or reference to directly invoke the services of the CORBA object.

[0010] In one embodiment an enterprise naming service for applications to locate services is provided. The enterprise naming service for applications to locate services comprises a binding module to associate an interface maintaining a reference to a first service with a location of the interface, the binding module further operable to associate a second service with a location of the second service and a look-up module to provide the location of the interface in response to a request by an application, the look-up module further operable to provide the location of the second service in response to a request by a second application. For example, in one embodiment of the naming service for applications to locate services, the binder module associates a java naming and

directory interface service maintaining an address or universal reference locator of an enterprise java bean with the address or universal reference locator of the java naming and directory interface service, and the look-up module provides the address or universal reference locator of the java naming and directory interface service in response to a request by an application for information on how to access the enterprise java bean. In another example, in one embodiment the binder module is operable to associate the name of a CORBA object to an address or reference to the CORBA object, and the name look-up module is operable to use the association to provide the address or reference of the CORBA object to an application which has requested the look-up information of the CORBA object. The application may then use this CORBA object address or reference to directly invoke the services of the CORBA object.

[0011] In still another embodiment, a method for locating a service in an enterprise is provided. The method comprises associating a service with a location with an interface maintaining a reference to a service. The method includes requesting, by an application desiring to employ the service, the location of the service, and returning the location of the interface to the application.

[0012] These and other features and advantages will be more clearly understood from the following detailed description taken in conjunction with the accompanying drawings and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] For a more complete understanding of the present disclosure and the advantages thereof, reference is now made to the following brief description, taken in connection with the accompanying drawings and detailed description, wherein like reference numerals represent like parts.

[0014] Figure 1 is a block diagram of an enterprise name service system according to one embodiment.

[0015] Figure 2 is a block diagram of an enterprise name service system according to another embodiment.

[0016] Figure 3 is a block diagram representing the enterprise name service as a series of cooperating layers.

[0017] Figure 4 is a flow chart for a method of looking up services in an enterprise name service system.

[0018] Figure 5A illustrates a sequence diagram of messages employed to access a CORBA based object through the enterprise name service system.

[0019] Figure 5B illustrates a sequence diagram of messages employed to access an enterprise java bean object through the enterprise name service system.

[0020] Figure 5C illustrates a sequence diagram of another method of employing messages through the enterprise name service system.

[0021] Figure 6 illustrates an application sharing the roles of both a service provider and a client application.

[0022] Figure 7 illustrates an exemplary general purpose computer system suitable for implementing the several embodiments of the enterprise name service system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0023] It should be understood at the outset that although an exemplary implementation of one embodiment of the present disclosure is illustrated below, the present system may be implemented using any number of techniques, whether currently known or in existence. The present disclosure should in no way be limited to the exemplary implementations, drawings, and techniques illustrated below, including the exemplary design and implementation illustrated and described herein.

[0024] Turning now to Figure 1 a block diagram of an enterprise name service (ENS) system 10 is depicted. Service providers 12a, 12b, and 12c are computer programs or applications which provide services to client applications 14a, 14b, and 14c. In some cases the service providers 12 may be interfaces which provide a mechanism for finding service objects, for example enterprise java bean objects, which may fulfill the request of the client application 14. The ENS system 10 provides service location transparency, for example, the service providers 12 may be relocated, and the client application 14 need not change its behavior when employing the ENS system 10 to find the service objects.

[0025] Stated in another way, of particular relevance to the present disclosure is the potential of addressing a number of different environments or domains within an enterprise in an integrated and cost effective manner. An embodiment of the present disclosure provides a single name look-up service which provides access to name look-up

for some domains directly, while for other domains provides a response to the name look up which provides a reference to the interface of a local name service for that domain. In this manner, efficiency may be gained by allowing clients throughout the enterprise to go to a single name service regardless of domain (providing the desired transparency), while at the same time reducing the need to recreate every potential name look-up for every domain in one omnibus application. One specific example of this approach is demonstrated below as object references in the CORBA domain are provided directly, while name look-ups in a JAVA domain are provided a reference to the interface of a java naming and directory service having appropriate references for the desired name look-ups in the desired JAVA domain.

[0026] In some cases, the client applications 14 access the services supported by the service providers 12 by invoking methods or function calls of application programming interfaces (APIs) provided by the service providers 12. In order to invoke these methods or function calls the client application 14 may need to know the location or address of the service provider 12. The ENS system 10 provides a mechanism for client applications 14 to look up the location or address of the service provider 12 at the time the client application 14 wishes to access the service supported by the service provider 12.

[0027] An enterprise name service (ENS) 16 comprises a binder module 18, a name look-up module 20, and a service look-up module 22. When brought into service, for example on start-up, the service provider 12 registers information with the enterprise naming service 16. This may be termed binding. This information may be an address or a location, a name, and a service type of the service provider 12. The binder module 18

creates a mapping which associates together the location, name, and service type of the service provider 12. This association or mapping is termed a binding, for example the name of the service provider 12 is said to be bound to the location and the service type of the service provider 12 via this association, mapping, or binding. The binding is stored in a datastore 24 which is accessible to the binder module 18, the name look-up module 20, and the service look-up module 22.

[0028] The service providers 12, the client applications 14, and the ENS 16 are computer programs or applications which may execute on a general purpose computer system. General purpose computer systems are discussed further hereinafter.

[0029] If the service provider 12 is relocated to execute on a different computer such that the old binding of the name of the service provider 12 with the location and the service type of the service provider 12 becomes invalid, the service provider 12 should rebind. Rebinding may involve deleting the former mapping or binding created for the service provider 12 and then creating a new mapping or binding with the new location of the service provider 12. Alternately, rebinding may involve revising the former mapping or binding.

[0030] When the service provider 12 is taken out of service, for example on shut-down, the service provider 12 unbinds its services. Unbinding is the operation of removing the effect of binding or registering. When the service provider 12 unbinds, the binder module 18 removes or marks as invalid the mappings or bindings created earlier.

[0031] It is the responsibility of the service providers 12 to keep their bindings current and up-to-date. Binding and rebinding is accomplished by the service providers 12 invoking functions provided by an ENS API.

[0032] Binding information which is provided by service providers 12 may vary substantially among different service types. Some example bindings are provided below. An enterprise java bean (EJB) binding may comprise a service name, a type, a version, a provider universal reference locator, an initial context factory, a java name and directory interface (JNDI) name, and a full home interface. An example EJB binding is:

```
servicename=EJBServicePlanManager  
type=EJB  
version=1.0  
provider_url=t3://localhost:9631  
initial_context_factory=weblogic.jndi.WLInitialContextFactory  
jndi_name=ExamplePlanManager  
full_home_interface=com.acme.enterprise.example.PlanManagerHome
```

[0033] A java messaging service (JMS) queue connection factory (QCF) binding may comprise a service name, a type, a version, a queue manager, a hostname, a port, and a channel. An example JMS QCF binding is:

```
servicename=QCF_EMER2DAV  
type=JMS  
version=1.0
```

qmgr=EMR2DAV

hostname=205.50.183.65

port=4231

channel=SYSTEM.DEF.SVRCONN

[0034] A JMS queue binding may comprise a service name, a type, a version, a queue manager, a queue, and a target client. An example JMS queue binding is:

servicename=Q_CLIENT

type=JMS

version=1.0

qmgr=EMR2DAV

queue=CLIENT

target_client=MQ

[0035] A java messaging service (JMS) Topic connection factory (TCF) binding may comprise a service name, a type, a version, a queue manager, a broker queue manager, a hostname, a port, and a channel. An example JMS TCF binding is:

servicename=TCF_EMR3DAV

type=JMS_TCF

version=1.0

qmgr=EMR3DAV

brkmgr=EMR3DAV

hostname=205.50.183.66

port=4232

channel=SYSTEM.DEF.SVRCONN

[0036] A JMS queue binding may comprise a service name, a type, a version, and a topic name. An example JMS Topic binding is:

servicename=T_CLIENT

type=JMS_T

version=1.0

tname=CLIENT

[0037] Note that the above bindings are only exemplary. In some embodiments other bindings and other service types may be employed.

[0038] When invoking a service or interface supported by the service provider 12 the client application 14 looks-up the service provider 12 through the ENS 16 by invoking functions provided by the ENS API. The client application 14 may look-up the service provider 12 by name. In this case, the client application 14 provides the name of the service provider 12 to the name look-up module 20, and the name look-up module 20 returns the location of the service provider 12. The name look-up module 20 employs the name provided by the client application 14 to search in the datastore 24 for the mapping of this name to the location of the service provider 12.

[0039] The client application 14 may also look-up the service provider 12 by service type. This is sometimes referred to as a trading service, but for the purposes of this application may also be referred to more generally as a naming service, as the deliverables are still names, it is the way in which they are requested which is distinct. In this case, the client application 14 provides the type of service provider 12 to the service look-up module 22, and the service look-up module 22 provides the names of all service providers 12 whose service type matches the service type provided by the client application. The client application 14 may then employ the one or more names returned by the service look-up module 22, by performing a look-up by name via the enterprise naming service 16, to obtain locations of the service providers 12 associated with the names. In some embodiments, the service look-up module 22 may provide both the names and locations of all service providers 12 whose service type matches the service type provided by the client application 14, thus saving the step of having to make a series of name look-ups via the name look-up module 20 to obtain locations of the service providers 12.

[0040] The client application 14 may employ criteria to select one of several service providers 12 identified through the service look-up. In some embodiments, the service look-up module 22 may employ a criteria, for example selecting the least recently used service provider 12, to select a single service provider 12 and to return the location or address of this single service provider 12 to the client application 14. For example, the service look-up module 22 may employ some algorithm to distribute processing loads evenly among several service providers 12 supporting the same service type.

[0041] The service providers 12 need to bind when they initialize and to rebind when they relocate. The client applications 14 need to look-up the location of the service provider 12 every time they access a service or interface supported by the service provider 12.

[0042] The ENS system 10 supports service location transparency for the client applications 14. The service provider 12 may be relocated arbitrarily, and the client application 14 need not change its behavior when employing this ENS system 10. The ENS system 10 is a unified naming service, for example applications 14 needing access to multiple service providers 12 need resort only to the single ENS system 10 to obtain the information necessary to access the service providers 12. By contrast, for example, if multiple java 2000 enterprise edition (J2EE) interfaces were to be accessed by a client application 14, the client application would need to know the universal reference locator (URL) of each java naming and directory interface (JNDI) associated with each separate J2EE interface since there is no unified J2EE repository for names at this time.

[0043] In some embodiments the ENS system 10 supports versioning of service providers 12. In this case, the ENS system 10 can support running multiple versions of service providers 12 in the same system and permits client applications 14 to select the version of service provider 12 desired. Recall that some service providers 12 may be interfaces which provide a mechanism for finding service objects. When versioning is supported, the service provider 12 provides version information when binding and rebinding, and client applications 14 may specify a version identifier when looking-up the service provider 12. When the client application 14 omits the version identifier when

looking-up the service provider 12, the name look-up module 20 returns the service provider 12 with the latest version identifier which matches the name.

[0044] Similarly, the client application 14 may specify a version identifier when looking-up by service type, and the service look-up module 22 returns all service providers 12 having the specified version and having the specified service type. If the client application 14 omits the version identifier when looking-up by service type, the service look-up module 22 returns all service providers 12 at the latest version and having the specified service type.

[0045] The communication protocol or object access mechanism employed by the client application 14 to access the services or interfaces provided by the service provider 12 determines which of several APIs supported by the ENS 16 that the client application 14 employs. CORBA objects are bound and looked-up via a CORBA COS naming interface. Java message queue (MQ) objects and java 2000 enterprise edition (J2EE) interfaces are bound and looked-up via a java ENS API. WebServices are bound and looked-up via a universal description discovery and integration (UDDI) ENS API. For example, if the client application 14 needs to access an object made accessible to it via CORBA, the client application 14 interacts with the ENS 16 via the CORBA COS naming interface. As a second example, if the client application 14 needs to access an enterprise java bean (EJB) object, the client application 14 interacts with the ENS 16 via the java ENS API.

[0046] When a CORBA object is requested, the ENS system 10 returns the object reference. With this object reference, the client application 14 may invoke the

needed service directly on the object. The service version concept described above may be applied to CORBA objects as well. In some embodiments, a service provider 12 may provide object version information when binding and rebinding its CORBA objects, and the client application 14 may specify object version information when looking-up CORBA objects.

[0047] For requests for other non-CORBA services, the ENS system 10 returns meta-information which allows the client application 14 to communicate directly with the service provider 12. In some cases, helper classes may be supported to make accessing the services or interface of the service provider 12 more convenient.

[0048] Turning now to Figure 2, another embodiment of the ENS system 10 is depicted. A lightweight directory protocol (LDAP) directory service is employed for accessing and organizing the datastore 24 in this embodiment. LDAP supports faster reads than writes, and in the ENS system 10 there should be more read operations than write operations. A LDAP server 26 communicates with the ENS 16 via LDAP protocol. A LDAP administration server 28 interacts with the LDAP datastore 24 and provides a central console accessible to a web administrator 30. The LDAP administration server 28 may be software provided by the vendor who may supply the LDAP store 24. The LDAP administration server provides an administrative interface to maintain the LDAP store directly. This may be useful to manually change entries and to perform occasional manual clean-up. For example, sometimes it may be necessary to delete stale or outdated references in the LDAP store 24 if a service provider 12 crashes and is not able to unbind

its services. In other embodiments, the datastore 24 may be a relational database or other database structure rather than a LDAP based datastore.

[0049] A name service browser 32 supports a hypertext markup language (HTML) world wide web interface that allows access by client applications 14 or service providers 12 to obtain a variety of service status information. The Name service browser 32 is capable of displaying a CORBA web interface and a Java web interface. The CORBA web interface provides a list of bound services and allows client applications 14 or service providers 12 to further explore and look into the details, as well as to ping the status of those services. The Java web interface provides the capability to search services by service name, version, and other attributes. The name service browser 32 is a useful tool for application administrators and service providers who bind the services, as well as clients who look-up the services. For example, the name service browser 32 provides a variety of useful functionality including the ability to be queried for available EJB services, querying if a specified service is running, identifying all or some server instances, and determining where a service is located, for example. The client application 14 or service provider 12 sends a request message to the name service browser 32, the name service browser 32 performs operations to obtain the requested information, and the name service browser 32 returns the information to the requestor.

[0050] Turning now to Figure 3, the ENS system 10 is depicted as a layered stack of intercommunicating modules. An API layer 34 comprises the CORBA COS naming API 36, the java ENS API 38, the UDDI ENS API 40, and the HTML interface. Service providers 12 bind and rebind their services and client applications 14 look-up

services using one of these three APIs 36, 38, 40. Service providers 12 and client applications 14 may send a query to the name service browser 32 through the HTML interface. A service layer 42 comprises the binder module 18, the name look-up module 20, the service look-up module 22, and the name service browser 32. The API layer 34 invokes the services of the service layer 42 to complete the API functions or methods invoked by the service providers 12 and client applications 14. A LDAP layer 44 provides LDAP services. The service layer 42 invokes the services of the LDAP layer 44 to complete the functions or methods invoked by the API layer 34. A datastore layer 46 provides datastore services. The LDAP layer 44 interacts with the datastore layer 46 to complete the functions or methods invoked by the service layer 42.

[0051] Turning to Figure 4, a flow chart of a service provider 12 location look-up is depicted. The process begins at block 50 and proceeds to block 52 where a service provider 12 provides a binding for its services. This involves providing a name, a location or address, and a service type. In some embodiments this also involves providing version information. The name, location or address, service type, and, optionally, the version information, are associated together in a map or a binding.

[0052] The process proceeds to block 54 where a client application 14 requests the look-up of the service location. The process proceeds to block 56 where a decision is made. If this is a name look-up, the process proceeds to block 58 where the map associated with the specified name is searched for and found. The process proceeds to block 60 where the map information is employed by the requesting client application 14 to

invoke the API of the service provider 12. The process proceeds to block 62 where the process exits.

[0053] If at block 56 a service type look-up is requested, the process proceeds to block 64 where the maps associated with the specified service type are searched for and found. The process proceeds to block 66 where one map from potentially many maps associated with the specified service type is selected. The process proceeds to step 60 where the selected map information is employed by the requesting client application 14 to invoke the API of the service provider 12. The process proceeds to block 62 where the process exits.

[0054] Turning now to Figure 5A, a message sequence diagram illustrates a typical CORBA object access using the ENS system 10 CORBA COS naming interface. The client application 14 sends a lookupObject message 80 to the ENS system 10. The ENS system 10 looks-up the specified object and returns a reference to the object in a locateObject message 82 to the client application 14. The client application 14 sends an invokeService message 84 to the object, referred to as a service object 86, using the reference to the object. Note that sending a message is synonymous with invoking a function or method.

[0055] Turning now to Figure 5B, a message sequence diagram illustrates a typical EJB object access using the ENS system 10. The client application 14 sends a lookupInterface message 100 to the ENS system 10. The ENS system 10 looks-up the specified interface and returns metadata in a locateInterface message 102 to the client application 14. The client application 14 employs the metadata to determine how to

access the interface, here an EJB service 104, and sends a lookupObject message 80 to the EJB service 104. The EJB service 104 looks-up the specified object and returns information necessary to communicate with the object via a locateObject message 82 to the client application 14. The client application 14 sends an invokeService message 84 to the EJB 106. Note that the message names above are exemplary and in practice other names may be used or other variant message sequences may be used.

[0056] Turning now to Figure 5C, a message sequence diagram illustrates a typical Web Service and other Business Service access using the UDDI ENS system 10. The client application 14 sends a lookup Business Detail message 80 to the ENS system 10. The ENS system 10 looks-up the specified Business Detail and returns the information on the Business Details in a locate Business Detail message 82 to the client application 14. The client application 14 sends an invoke Business Service message 84 to invoke the Business Service, referred to as a business service 86. Note that sending a message is synonymous with invoking a function or method.

[0057] The above three described sequence diagrams illustrate the service location transparency supported by the ENS system 10. The location of the service provider 12 may be changed without effecting the behavior of the client applications 14.

[0058] Note that some applications may act in the role of service provider 12 relative to one application but act in the role of a client application 14 relative to another application. Turning to figure 6, for example, application T 140 may request a service named serviceS by sending a requestServiceS message 142 to application S 144, application S 144 may perform serviceS, and application S 144 may send a

serviceSPerformed message 146 to application T 140 to satisfy the request for serviceS. In this example application S 144 acts as a service provider 12 relative to application T 140 which acts as a client application 14. At the same time, application S 144 may request a service named serviceR by sending a requestServiceR message 148 to application R 150, application R 150 may perform serviceR, and application R 150 may send a serviceRPerformed message 152 to application S 144 to satisfy the request for serviceR. In this example application S 144 acts as a client application 14 relative to application R 150 which acts as a service provider 12.

[0059] The ENS system 10 described above may be implemented on any general-purpose computer with sufficient processing power, memory resources, and network throughput capability to handle the necessary workload placed upon it. Figure 7 illustrates a typical, general-purpose computer system suitable for implementing one or more embodiments disclosed herein. The computer system 380 includes a processor 382 (which may be referred to as a central processor unit or CPU) that is in communication with memory devices including secondary storage 384, read only memory (ROM) 386, random access memory (RAM) 388, input/output (I/O) 390 devices, and network connectivity devices 392. The processor may be implemented as one or more CPU chips.

[0060] The secondary storage 384 is typically comprised of one or more disk drives or tape drives and is used for non-volatile storage of data and as an over-flow data storage device if RAM 388 is not large enough to hold all working data. Secondary storage 384 may be used to store programs which are loaded into RAM 388 when such programs are selected for execution. The ROM 386 is used to store instructions and perhaps data

which are read during program execution. ROM 386 is a non-volatile memory device which typically has a small memory capacity relative to the larger memory capacity of secondary storage. The RAM 388 is used to store volatile data and perhaps to store instructions. Access to both ROM 386 and RAM 388 is typically faster than to secondary storage 384.

[0061] I/O 390 devices may include printers, video monitors, keyboards, mice, track balls, voice recognizers, card readers, paper tape readers, or other well-known input devices. The network connectivity devices 392 may take the form of modems, modem banks, ethernet cards, token ring cards, fiber distributed data interface (FDDI) cards, and other well-known network devices. These network connectivity 392 devices may enable the processor 382 to communicate with an Internet or one or more intranets. With such a network connection, it is contemplated that the processor 382 might receive information from the network, or might output information to the network in the course of performing the above-described method steps. Such information, which is often represented as a sequence of instructions to be executed using processor 382, may be received from and outputted to the network, for example, in the form of a computer data signal embodied in a carrier wave.

[0062] The processor 382 executes instructions, codes, computer programs, scripts which it accesses from hard disk, floppy disk, optical disk (these various disk based systems may all be considered secondary storage 384), ROM 386, RAM 388, or the network connectivity devices 392.

[0063] In one embodiment the ENS system 10 described above is flexible and extensible to existing and emerging technologies, and supports service provider 12 location transparency, enabling easy relocation of service providers 12 to distribute loads across computer systems.

[0064] While several embodiments have been provided in the present disclosure, it should be understood that the disclosed systems and methods may be embodied in many other specific forms without departing from the spirit or scope of the present disclosure. The present examples are to be considered as illustrative and not restrictive, and the intention is not to be limited to the details given herein, but may be modified within the scope of the appended claims along with their full scope of equivalents. For example, the various elements or components may be combined or integrated in another system or certain features may be omitted, or not implemented.

[0065] Also, techniques, systems, subsystems and methods described and illustrated in the various embodiments as discreet or separate may be combined or integrated with other systems, modules, techniques, or methods without departing from the scope of the present disclosure. Other items shown as directly coupled or communicating with each other may be coupled through some interface or device, such that the items may no longer be considered directly coupled to each but may still be indirectly coupled and in communication with one another. Other examples of changes, substitutions, and alterations are ascertainable by one skilled in the art and could be made without departing from the spirit and scope disclosed herein.